Preprint
UCRL-JC-153559

Achieving Order through CHAOS: the LLNL HPC Linux Cluster Experience

R.L. Braby, J.E. Garlick, and R.J. Goldstone

This article was submitted to
ClusterWorld, San Jose, CA, June 23–26, 2003

U.S. Department of Energy

Lawrence
Livermore
National
Laboratory

May 2, 2003

DISCLAIMER

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

This is a preprint of a paper intended for publication in a journal or proceedings. Since changes may be made before publication, this preprint is made available with the understanding that it will not be cited or reproduced without permission of the author.

Achieving Order through CHAOS: the LLNL HPC Linux Cluster Experience

Ryan L. Braby, Jim E. Garlick, and Robin J. Goldstone
Production Linux Group
Lawrence Livermore National Laboratory, USA

Abstract

Since fall 2001, Livermore Computing at Lawrence Livermore National Laboratory has deployed 11 Intel IA-32–based Linux clusters ranging in size up to 1154 nodes. All provide a common programming model and implement a similar cluster architecture. Hardware components are carefully selected for performance, usability, manageability, and reliability and are then integrated and supported using a strategy that evolved from practical experience. Livermore Computing Linux clusters run a common software environment that is developed and maintained in-house while drawing components and additional support from the open source community and industrial partnerships. The environment is based on Red Hat Linux and adds kernel modifications, cluster system management, monitoring and failure detection, resource management, authentication and access control, development environment, and parallel file system. The overall strategy has been successful and demonstrates that world-class high-performance computing resources can be built and maintained using commodity off-the-shelf hardware and open source software.

1 Introduction

For the past several years, Lawrence Livermore National Laboratory (LLNL) has invested significant efforts[1] in the deployment of large high-performance computing (HPC) Linux clusters and in building system software and strategies to manage them. After deploying two modest-sized clusters, the Parallel Capacity

Resource (PCR) systems (Emperor, 88 nodes; Adelie, 128 nodes) in fall 2001, these efforts progressed to the successful deployment of the Multiprogrammatic Capability Resource (MCR, 1154 nodes) in fall 2002 and the ASCI Linux Cluster (ALC, 962 nodes) in winter 2003. In November 2002, MCR achieved a LINPACK result of 7.642 TFLOPS, placing it fifth on the 21st TOP500 Supercomputer list (http://www.top500.org/list/2002/11/). This was accomplished with a total system cost (hardware including maintenance, interconnect, global file system storage) of under $14 million.

These systems are all running customer applications and are in various stages of production readiness, with PCR in "general availability" (full production) and MCR and ALC in "science run" mode (users expect to have to work around some system limitations). LC systems provide a common programming model to users, which minimizes the amount of work necessary for them to get applications running on new platforms. Section 2 describes the programming model as implemented by LC Linux systems, and Section 3 describes the Linux cluster architecture used to implement the model.

Livermore has experimented with various system integration techniques for Linux clusters. Selecting the right hardware, determining the best integration approach, and deciding on a hardware support model are discussed in Section 4.

System software has traditionally been a problem area for Linux clusters. Livermore's strategy is to leverage partnerships with industry and the open source community to build and support various software components and to put the components together into a Red Hat-based, in-house Linux distribution called CHAOS (Clustered High Availability Operating System)[2]. This strategy is detailed in Section 5.

While implementing and supporting HPC Linux clusters, LLNL staff have learned many lessons the hard way. Section 6 describes some of these lessons. Section 7 summarizes LLNL's experience with the aforementioned clusters and evaluates the current state of Linux clusters at Livermore, and Section 8 presents LLNL's path forward for HPC Linux clusters.

2 Background

LLNL has been at the forefront of HPC for many years. LC provides the majority of unclassified and classified computing cycles to scientific programs throughout LLNL. In recent years, clusters of SMP nodes have provided the bulk of those cycles. As part of the ASCI Program, LC has deployed two generations of IBM SP ranking first and second on the TOP500 list.[1] Recent successes with Linux clusters mostly built from commodity off-the-shelf (COTS) components indicate that a large portion of our customers' computing requirements can be served with these cost-effective, stable, and easily manageable systems.

[1] ASCI Blue-Pacific SST was second on the 14th TOP500 Supercomputer Sites list: http://www.top500.org/lists/1999/11/, and ASCI White was first on the 16th list: http://www.top500.org/lists/2000/11/ (as well as the 17th and 18th lists).

LC clusters fall into two architectural categories: *capability* clusters, which can run large-scale, distributed-memory applications and implement the Livermore Model[3], described below; and *capacity* clusters, which are a loosely coupled collection of systems that run a serial workload. Linux clusters of both architectures have been deployed at Livermore.

LC's scalable systems strategy for capability systems, known as the Livermore Model (see Figure 1), is to provide a common application programming and execution framework. This allows complex scientific simulation applications to be portable across multiple platforms and provides a stable target environment over multiple generations of platforms. This strategy has been successful since about 1992, when the Meiko CS-2 MPP was introduced at LLNL.

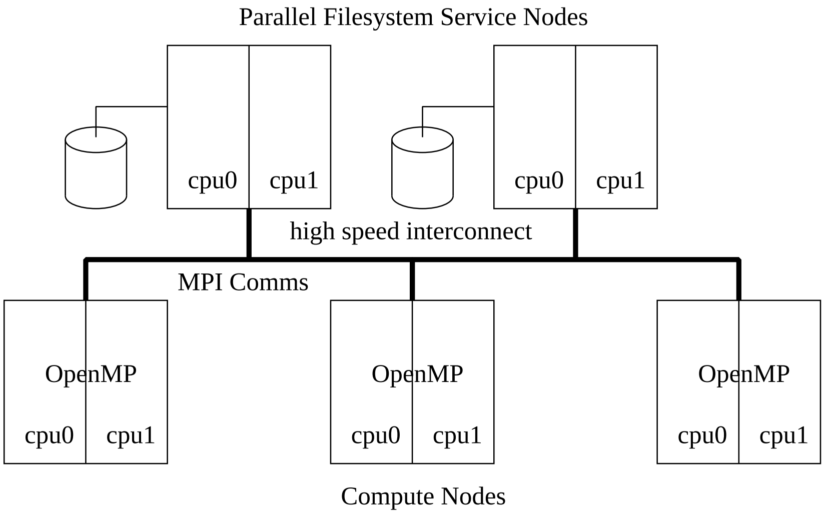

Fig. 1. The Livermore Model

The Livermore Model seeks to abstract the parallelism and I/O services of the computing environment to a high level that evolves slowly. The abstraction is based on SMP compute nodes attached to a high-speed, low-latency message passing interconnect. Applications utilize compiler-generated OpenMP threads to exploit SMP parallelism and MPI to exploit parallelism between nodes. Data sets are stored on a common (POSIX interface) parallel file system, which may utilize striping across multiple file service nodes in order to achieve the required bandwidth and scalability.

In addition to the programming model abstraction, the Livermore Model assumes parallel cluster management tools, resource management and control with near-real-time accounting, job scheduling for allocation management, C, C++, and

Fortran compilers, scalable MPI/OpenMP GUI debugger, and performance analysis tools.

Capacity systems, on the other hand, are oriented toward a serial workload; thus, they do not provide the message passing abstraction between nodes or parallel I/O capability, and they require a different resource management strategy. They do, however, provide the same serial tools (compilers, for example) and have much in common with capability systems from a system management perspective.

3 Cluster Architecture

Minimizing architectural differences between clusters enables support personnel to move between clusters with less confusion and simplifies software testing and configuration. All LC Linux clusters (both capability and capacity) share a common cluster architecture (described in detail below) that includes one or two dedicated "management" nodes, a private management IP network, a serial console/remote power management infrastructure, a public IP network that makes nodes externally visible, and one or more dedicated "login" nodes. Capability systems have additional common architectural traits required to implement the Livermore Model, including high-speed, message passing interconnect and storage architecture for parallel I/O. Figure 2 depicts the 1154-node MCR cluster, illustrating the architecture typical of LC capability Linux clusters.

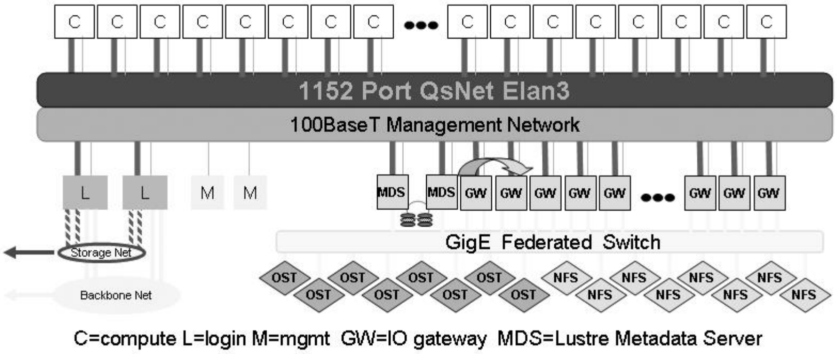
C C C C C C C C
•••
C C C C C C C C
1152 Port QsNet Elan3
100BaseT Management Network
L L M M
MDS MDS GW GW GW GW GW
•••
GW GW GW
Storage Net
Backbone Net
GigE Federated Switch
OST OST OST OST OST OST OST OST
NFS NFS NFS NFS NFS NFS NFS NFS
C=compute L=login M=mgmt GW=IO gateway MDS=Lustre Metadata Server

Fig. 2. MCR architecture

A dedicated management node serves as the access point for system administration of the cluster and provides compute cycles for critical system processes such as the resource manager, power/console management daemons, Simple Network Management Protocol (SNMP) monitoring, and remote OS installation. On larger clusters, a second management node is used to provide additional capacity and fail-over capability. Pluggable Authentication Module (PAM) access controls limit access to support personnel only. Management nodes are attached to the private management network via gigabit Ethernet (GigE).

All nodes in the cluster are attached to a private (192.168.x.x or 10.x.x.x), switched management Ethernet. This network carries system traffic such as DNS, SNMP polls and traps, ganglia multicast, and resource manager heartbeats, some of which may be sensitive to changes in quality of service and therefore must be partitioned away from unpredictable, application-generated network activity. The management network is also used for remote OS installation, software updates, and system administrator access to nodes, although this type of activity is usually limited to maintenance windows when impact on the aforementioned services is less critical. Application- or user-generated network activity is mostly excluded from this network by virtue of having node host names associated with another “public” interface.

Because it is a practical necessity when deploying a large cluster to be able to manage nodes remotely, all nodes have serial consoles attached to terminal servers and power plugs attached to remote power controllers. Software on the management nodes logs console activity, brokers interactive console sessions, and controls power. Power controllers and terminal servers are either attached to the private management network or to a separate dedicated network with connectivity to the management nodes, depending on security capabilities of the specific hardware used and the cluster size.

LC Linux clusters are deployed in a “service rich” environment having large center-wide NFS servers, authentication services, automated backup services, etc. It is convenient to give all nodes in the cluster network routes to reach these services so applications can make use of them.[2] On clusters with a message passing interconnect that can pass IP traffic, the interconnect carries external network traffic, with “login” nodes or other dedicated “gateway” nodes acting as routers. On other clusters, one or more Ethernet networks are used for this purpose, and a regular router is used. The primary host name of each node corresponds to its interface on this public network.

Several nodes in the cluster are designated as “login nodes” and serve as the interactive access point for the cluster. Typically, each cluster is given a convenient DNS alias that is configured to cycle among the login nodes for a primitive load balancing effect. Login nodes are a place for development and compilation of applications, batch job submission, workload monitoring, and review of results. Users typically offload results to the LLNL archive or other systems for post-processing using a number of dedicated, high-speed network interfaces[3] installed in the login nodes. As mentioned above, login nodes may also be used as routers for the public network when it is implemented on a non-commodity medium that cannot be attached directly to a commercial router.

[2] LC customers have come to expect that center-wide NFS home directories and other common file systems are mounted on every node of every cluster; however, they are discouraged from performing parallel I/O on these file systems for obvious reasons.

[3] Archival tools such as Parallel FTP can stripe data across multiple interfaces.

A high-speed interconnect is used for message passing, for bulk data movement for a parallel file system, and to carry IP traffic for the public network. When configured in a cluster, the interconnect is normally attached to all nodes except the dedicated management nodes. Interconnect switches may be remotely manageable, in which case they are attached either to the management network or the power/console network (because they are normally accessed only by the management nodes).

Considerable research and development is being carried out at Livermore and elsewhere in the area of parallel file systems; thus, there are two storage architecture approaches currently in use. Both approaches dedicate several nodes as I/O servers, over which the rest of the cluster stripes data transfers to achieve the required parallel I/O bandwidth. In one approach, storage is directly attached to the I/O nodes via FibreChannel. In the other, the I/O nodes act as gateways for GigE-attached storage.

4 Hardware Choices and Integration Strategy

LC invests significant effort in ensuring that its computer systems meet established high standards of performance, usability, manageability, and reliability. For the largest systems, a platform architect is responsible for the overall system design. A Request for Proposal (RFP) is drafted and published, and responses are evaluated against the often large number of requirements set out in the RFP. A vendor is selected and a Statement of Work (SOW) is written to further clarify the details of the configuration, delivery schedule, integration requirements, and acceptance criteria. The process, from beginning to end, typically takes about 6 months. For smaller clusters, some of these steps can be skipped, although in all cases much attention is given to the configuration details that will ensure a successful deployment.

As an RFP is prepared, new hardware is evaluated in LC's I/O Testbed facility[4] to ensure that requirements are achievable and to begin to solidify some details of the procurement, such as what adapters will be required and what memory and interconnect bandwidth should be demonstrable. In addition, for every large production cluster procured, a small test cluster of the same architecture is deployed in the testbed, where it is used to test new software and system configurations before they are tried on the production platforms. The testbed cluster is usually requested to be shipped to Livermore as early as possible so that LC can work in parallel with the vendor toward solving the inevitable problems that will delay acceptance.

4.1 Hardware Choices

When building Linux clusters with COTS components, it is important to recognize that specifying components requires a much greater level of care than selecting

[4] The I/O Testbed facility is so named because of its origins as a testbed for LC's storage archive.

components to build a desktop PC. Motherboard chip sets need kernel support for environmental sensor access and memory error detection. Hardware incompatibilities may arise between the BIOS and high-performance peripheral adapters. Hardware that is well supported under Windows might not perform as well, be as stable, or work at all under Linux. Subtle differences in hardware can manifest themselves as significant differences in the overall manageability and mean time between failure (MTBF) of a large cluster. It is therefore important to be clear and detailed with regard to the functionality and performance required of new hardware in an RFP and to test as much as possible in advance of entering into a binding agreement with a vendor.

Hardware considerations fall into four broad categories: nodes, interconnect, storage, and remote manageability.

Nodes LC clusters have several types of service nodes (e.g., "management," "login," "io," and in some cases other types) and a single type of "compute" node. Because there are considerably fewer service nodes than compute nodes, compact form factor is more critical in compute nodes, while the number of I/O slots and I/O bus bandwidth tend to be more important in service nodes. The rest of the node requirements are usually the same across all types. Node requirements can be divided into six categories: processor technology, memory, motherboard, internal disk, PCI configuration, and form factor.

Assuming that the processor technology is not the first of an architecture to be deployed,[5] the number of processors, the processor generation, and a minimum clock speed would usually be specified in the RFP. New revisions of commodity processors are released frequently, and vendors vary in their ability to obtain the latest revisions in quantity, so this is sometimes an area in which vendors can differentiate themselves in a competitive response.

Because memory is costly, the amount required will usually be determined in advance by the budget allotted for the procurement and will not vary between responses. On Intel IA-32 architecture systems, it is worth noting that there are software risks when memory exceeds two thresholds, one at 4 GB (where Intel's Physical Address Extension (PAE) mode has to be enabled to address the high memory) and one at approximately 8 GB (where the size of virtual memory–related kernel tables starts to become unwieldy). It should be noted that PCI address space is typically mapped into the top of the IA-32 4-GB physical address space, so the first threshold is somewhat less than 4 GB, depending on the PCI devices used. Also, there are potential issues with PAE mode. Not all chip sets support remapping of physical memory shadowed by PCI space to addresses above 4 GB (where it could be addressed in PAE mode), in which case the memory would be unusable. PAE mode also risks breaking kernel driver code that has not been rigorously tested

[5] Before a new architecture would be considered, extensive testing with user applications and internal LC planning would occur that is beyond the scope of this paper; suffice it to say that an RFP for a cluster of a new architecture would be very rigorous and would require prospective vendors to share some of the additional risk contractually.

on large memory systems; for instance, physical addresses will no longer fit in the 32-bit "unsigned long" data type. PAE mode effectively halves the size of the hardware translation look-aside buffer (TLB), which could affect performance on some workloads. So, while PAE mode allows configurations with more than 4 GB of memory, it introduces a number of potential issues. Some research and testing is appropriate before crafting requirements that exceed either of these thresholds. In addition to the amount of memory, the type (e.g., DDR-200 versus DDR-266), the stick denominations (leaving room for expandability if desired), and memory error checking and correction technology such as ChipKill[4] might be specified.

The choice of motherboard may affect performance as well as manageability. The motherboards deployed in LC Linux clusters are listed in Table 1. Memory bandwidth is dependent on front side bus (FSB) speed and supported memory technology, PCI capabilities and configurations vary, and ability to properly handle various error conditions such as memory[6] or PCI parity errors is chip set specific. A motherboard implies a particular BIOS, and BIOSes should be able to communicate over the serial console and be upgradeable and configurable from Linux. LinuxBIOS[5], an open source BIOS, provides these capabilities and was designed with clusters in mind; however, it requires a significant development effort to port and test for each new motherboard. For visualization applications, AGP support may be packaged with motherboards intended for desktop systems, but desktop motherboards may not fulfill other requirements, such as BIOS serial console redirection. Drivers must exist for motherboard sensors. Support is affected by public availability of chip set programming information. When it is unavailable, the ability of on-site programmers and the open source community to solve chip set–related problems will be diminished, so it becomes more important that the vendor be capable of providing this level of support in the BIOS and kernel.

Table 1. Motherboards deployed in LC Linux clusters

Motherboard	*Chip Set*	*Memory Type*	*FSB (MHz)*	*PCI Slots*	*AGP*	*BIOS*
SuperMicro P4DC6+	Intel 860	RDRAM	400	6 PCI	4x	Award (modified)
SuperMicro P4DPR-iGM	Intel E7500	DDR-200	400	1 PCI-X, 1 PCI	-	LinuxBIOS/Phoenix
SuperMicro P4DPE	Intel E7500	DDR-200	400	6 PCI-X	-	LinuxBIOS/Phoenix
IBM xSeries x335	ServerWorks GC/LE	DDR-200	400	2 PCI-X	-	IBM
IBM xSeries x345	ServerWorks GC/LE	DDR-200	400	4 PCI-X, 1 PCI	-	IBM
SuperMicro X5DP8-G2	Intel E7501	DDR-266	533	6 PCI-X	-	Phoenix

In LC Linux clusters, each node has a complete copy of the OS, a large user scratch area, and swap space configured on a local disk. IDE drives are preferred

[6] Some systems respond to an uncorrectable memory error through a BIOS handler for a System Management Interrupt (SMI). Others generate a non-maskable interrupt (NMI) that is merely logged on the console by Linux. Still others do nothing at all. A kernel module for polling various memory controllers for parity errors is discussed in Section 5.4

from a cost standpoint, and in most cases redundant system disks are not justified because of high MTBF and the ease with which these drives can be replaced and reinstalled. When redundancy is required, SCSI is used because of the availability of on-board SCSI RAID controllers and hot swap, front-panel access. Recently, a number of vendors have begun offering ATA RAID solutions that offer hot swap capability and an attractive price. This will likely replace SCSI RAID in future LC clusters. If hardware RAID is to be used, it is important to verify that failed disks can be detected through Linux. It may be appropriate to include minimum specifications for the local disk in an RFP, such as rotational speed and interface type (e.g., ATA100).

The PCI subsystem is important on nodes with high-speed storage, networking, or message passing adapters. Because interconnect performance is critical to overall cluster performance, minimum bandwidth and latency for the chosen interconnect should be specified in an RFP. Nodes packaged in compact enclosures may need riser cards and should accommodate the adapter form factor. Configurations for service nodes with multiple adapters should be specified in detail to ensure that the appropriate number of slots and amount of bus bandwidth is available to achieve performance targets.

Dense node packaging saves costly computer room floor space but affects power distribution, heat dissipation, serviceability, and data cabling. "Blade servers" promise to achieve high density while alleviating the serviceability and cabling concerns, but blade processor and chip set technologies lag those offered by COTS products, and blade densities preclude the use of the highest clock speeds. One rack unit (1U) seems to be an empirical limit for node density in HPC Linux clusters.[7] At 1U per node, or about 40 nodes per rack, each rack might require the dissipation of up to 5.2 kW[8] of heat from the processors alone. In addition to potentially taxing computer room air conditioning (CRAC) capacity, this much heat density creates air flow challenges.

Message Passing Interconnect Systems implementing the Livermore Model must provide a low-latency, high-bandwidth interconnect optimized for message passing. The interconnect is such a critical component, and is so closely linked to system software, that in all procurements of capability clusters thus far, LC has purchased the interconnect separately and provided it to the system integrator as government furnished equipment (GFE). Early in LLNL's Linux effort, a relationship with Quadrics, Ltd. (http://www.quadrics.com) of Bristol, U.K., was established to port device drivers for the Quadrics QsNet interconnect from Tru64 to

[7] Linux NetworX built a "0.8U" solution for MCR that vertically mounts ten dual-Xeon nodes in an 8U chassis, but because gaps must be left between chassis for air intake, it is effectively a 1U node density.

[8] Each rack would dissipate 5.2 kW if each node contains dual 2.4-GHz Intel Xeon processors, which have a thermal guideline of 65.0 W; http://processorfinder.intel.com/scripts/details.asp?sSpec=SL6K2.

Linux. This collaboration has been very beneficial over several years, and LC has exclusively used QsNet in its capability Linux clusters.

Message passing interconnects require significant supporting software to operate, including a resource manager, MPI implementation, and kernel device drivers. Quadrics provides open source MPI (based on MPICH), kernel device drivers, and user space libraries. Their resource management product (RMS) is not open source and is expected to be replaced on LC clusters with SLURM (Simple Linux Utility for Resource Management)[6], an open source resource manager developed at Livermore (see Section 5.5). In addition to application message passing, the interconnect carries NFS/IP traffic and does bulk data movement for the Lustre parallel file system on LC clusters.[9] Impacts on these uses must also be considered when specifying the interconnect.

QsNet achieves latency and bandwidth on the order of 5 μs and 320 MB/s between user processes on a pair of nodes. Results vary depending on motherboard chip set (see Table 2), so it is important that an RFP establish minimum interconnect performance requirements in conjunction with node requirements, discussed in Section 4.1.

Table 2. QsNet MPI ping-pong between a pair of nodes (mping)

Chip Set	*Min Latency* (μs)	*Max Bandwidth* (MB/s)
Intel 860	5.3	224
Intel E7500	5.0	322
Intel E7501	4.5	323
ServerWorks GC/LE	4.6	325

QsNet has two primary hardware components: the QM400 Elan3 64 bit/66 MHz PCI adapter and the 16x16 QM401 Elite switch module. For clusters of 16 nodes or fewer, a single QM401 is packaged in a standalone switch configuration. For clusters of 16 to 128 nodes, a backplane capable of holding eight QM401s and additional Elite stages to tie them together is available. For larger configurations, multiple 128-way chassis dedicate some number of ports to up-links and are tied together with trunk switches to form a *federated* QsNet switch. A 1024-way federated switch that implements a full fat-tree requires 8 trunk switches and 16 first level switches (each with 64 up-links and 64 down-links). A less scalable but more cost-effective 1536-way federated configuration requires 4 trunk switches and 16 first level switches (each with 32 up-links and 96 down-links). Livermore clusters have used all of these configurations except the 1024-way full fat tree.

While other message passing interconnect technologies exist, LC has chosen to stay focused on QsNet for its current generation of production Linux clusters.

[9] User applications tend to separate computation and file I/O, so this dual use has little impact on application performance.

Other technologies for future generations are being evaluated, including Infiniband and QsNet Elan4.

Storage LC's parallel file system effort, described in Section 5.7, is focused on Lustre[7]. Initially an intra-cluster file system similar to others commonly found on Linux clusters, Lustre will grow to a scalable, global, secure file system capable of serving the needs of an entire data center. The underlying storage systems chosen to support Lustre initially can be either direct-attached storage connected to cluster I/O nodes or network-attached storage (NAS) accessed by I/O nodes over a storage network.

The fundamental unit of storage service used by Lustre is called an Object Storage Target (OST). Direct-attached storage would run OST software under Linux on the I/O node, while NAS would have to implement OST protocol "natively." Because OSTs will eventually need to be accessed center-wide, LC has some bias toward building relationships with storage companies willing to implement the OST protocol in NAS devices, despite the fact that an OST embedded in a storage appliance is more difficult to debug than an OST running under Linux. Thus for strategic reasons, storage hardware has been omitted from cluster RFPs and has been provided to integrators as GFE.

Currently, LLNL is using two main types of storage for HPC Linux clusters: BlueArc NAS systems and Data Direct Networks (DDN) FibreChannel arrays. Both companies are working with LLNL and Cluster File Systems, Inc. (http://www.clusterfs.com) on native OST implementations.

The BlueArc NAS solution consists of BlueArc SiliconServer Si7500 servers (http://www.bluearc.com) with approximately 1.4 TB of disk per server. Each BlueArc is connected to a private GigE switched network (1 fibre GigE port). This GigE network is connected to the cluster I/O nodes (2 copper GigE ports). I/O traffic to and from the BlueArcs is routed through the I/O nodes to the rest of the cluster. On MCR, seen in Figure 2, 64 BlueArc servers are connected to the GigE federated switch. Some of these BlueArcs are serving NFS (until Lustre is ready for production use), while the rest are Lustre OSTs.

The DDN FibreChannel solution utilizes S2A8000 Storage Controllers (http://www.datadirectnetworks.com/products/san/s2a8000.html). Each of these controllers offers 8 full-duplex FC-2 host connections and 20 FC drive loops. Dual-port FC-2 host bus adapters installed in the I/O nodes are attached directly to two S2A8000 host connections. Because the configured logical unit numbers (LUNs) on the S2A8000s appear as standard SCSI block devices on the I/O nodes, they can be used either as storage backing Linux-based NFS service or Linux-based OST service.

Remote Power and Console Management The choice of terminal server and remote power controller (RPC) for a cluster is driven by compatibility with LC console and power management software, security requirements, and form factor. Table 3 lists the devices deployed in LC Linux clusters.

Table 3. Remote console/power control devices deployed in LC Linux clusters

<table>
<tr><th><i>Cluster</i></th><th><i>Node Count</i></th><th><i>Power Controller</i></th><th><i>Terminal Server</i></th></tr>
<tr><td>Adelie</td><td>128+1</td><td colspan="2">LNXI Ice Box V2</td></tr>
<tr><td>Emperor</td><td>96+1</td><td colspan="2">LNXI Ice Box V2</td></tr>
<tr><td>Pengra</td><td>16+1</td><td>BayTech RPC3</td><td>Cyclades TS1000</td></tr>
<tr><td>ILX</td><td>45+1</td><td>BayTech RPC3</td><td>Cyclades TS1000</td></tr>
<tr><td>MCR</td><td>1152+2</td><td colspan="2">LNXI Ice Box V3</td></tr>
<tr><td>ALC</td><td>960+2</td><td>IBM xSeries service processor</td><td>MRV iTouch IR-8040</td></tr>
<tr><td>PVC</td><td>70+1</td><td>BayTech RPC3</td><td>Cyclades TS1000/TS2000</td></tr>
</table>

Terminal servers must support “reverse Telnet,” in which each serial port on the server is directly accessed through a unique IP address:port pair. Units that instead provide a command-line interface with commands to connect through to the individual ports are not compatible with LC console management software, which maintains a persistent connection to each serial port for continuous logging and interactive access. Serial consoles should run with no handshaking (software or hardware) at the fastest baud rate possible without data loss. A rate of 38,400 baud is a reasonable minimum. Because the console serial driver operates in a polled mode and the length of time spent busy-waiting is inversely proportional to the baud rate, kernel routines that write to the console, especially from an interrupt handler, can create instabilities at lower baud rates. Handshaking would exacerbate this problem by making the length of time spent busy-waiting dependent on the terminal server’s load. As a general rule, baud rates should be set such that all ports can simultaneously send about 2 KB (the size of a long kernel “oops” message) without data loss.

RPCs provide either a serial interface that can be plugged into a terminal server serial port and accessed via Telnet or a network interface that supports Telnet directly. LC power control software uses a fairly primitive Expect-like scripting language to control the plugs, so line oriented interfaces are preferred over those that make use of ANSI terminal features. The interface should at least support “plug on,” “plug off,” and “plug status” commands. Software maintains a persistent connection to the remote power controllers to reduce command latency, so it should be possible to disable inactivity timeouts. On large clusters, devices that control multiple plugs are preferred over devices that control only one (such as an embedded node service processor with its own serial port or Ethernet connection), because this reduces the amount of terminal server/Ethernet switch infrastructure and cabling required and also keeps down the number of sockets that need to be held open by the power control software. Some RPCs provide the capability of driving the hardware reset pin on the motherboard as an alternative to power cycling a system. RPC devices with this capability are usually provided as part of an integrated solution, such as an embedded service processor or custom node packaging.

Some RPCs and terminal servers implement simple security measures such as host access lists. When access lists are unavailable, these devices must be attached to a network that is only reachable from nodes with restricted user access; otherwise, it may be possible for users to boot a node in single user mode and give themselves root access.

Form factor is a consideration in dense cluster designs. Terminal servers are available in port densities up to 48 ports in a single rack space (1U). RPC form factors are more numerous, including plug strips that mount vertically, consuming zero vertical rack space. There are also combination units such as the Linux NetworX ICE Box (http://www.lnxi.com), which provides RPC and terminal server functionality for ten nodes in a chassis and mounts in the rear of the rack behind the nodes. RPC selection should be part of an overall plan for cluster power distribution. Depending on the cluster density and RPC device, it may be appropriate to install a power distribution unit (PDU) in each rack to aggregate RPC supply connections into a single supply cable for the rack and to provide a local "master" power switch for the rack.

4.2 Integration Considerations

It may seem that building commodity Linux clusters is a simple matter of stacking a bunch of PCs in a rack. Yet this seemingly straightforward task is not simple. Many minor details need attention, and it can require a lot of time to assemble a collection of parts into a large cluster. When dealing with integration partners, it is important to have clear requirements for a cluster. While LC system administrators have successfully built and deployed clusters of 17, 46, and 71 nodes in size without an integrator, the process was tedious and not very cost effective. Unless inexpensive labor is available, anything larger than a single rack cluster is probably not suitable for self-integration. For these larger systems, some amount of vendor integration services are justified.

Regardless of whether or not a vendor partner is used for system integration, some details that should not be ignored include racks, cabling, labels, BIOS/CMOS settings, and hardware burn-in.

The cluster will need racks, and all racks are not created equally. A high-quality rack with flexible mounting options and generous space for cable routing will improve the quality of the finished product. The Wrightline (http://www.wrightline.com/) Vantage series of racks has been a good choice for LC installations.

Wherever possible, network switches, power controllers, and terminal servers should be located to minimize cable length and cable crossover between racks. Heavier components should be placed lower in the racks for stability and increased ease of service. For aesthetic and airflow purposes, blank spaces should be consolidated at the tops of racks, and blank filler plates should be used. Power switches should be covered by rack doors or otherwise located or protected so that they won't be accidentally turned off.

A rack full of tangled wires is an eyesore and a serviceability nightmare. Cable lengths should be optimized to avoid excessive coils of unneeded cables, and cables should be routed down the sides of the racks and tied off with Velcro bands.

Plastic zip ties should be avoided, because they make it difficult to reroute or replace cables. Cables should not be crimped or bent at excessive angles. DB-type cables should be screwed down to ensure good connections. RJ-type cables should use non-hooded connectors, because rubber hoods can impede the ability to make a secure connection and make it difficult to release in tight spaces such as dense Ethernet switches.

All cables in the cluster should be labeled at both ends with a source and destination identifier. This will aid in the debugging of any connection problems and ensure that components can be removed and replaced reliably. Labels should be of high quality to ensure that the ink does not smear and that they do not fall off as a result of excessive heat.

It is likely that some BIOS/CMOS settings will need to be adjusted to effectively use COTS hardware in a Linux cluster. For example, most PC BIOSes do not enable serial console redirection by default. When nodes are delivered from the vendor, BIOSes may not be at the current revision level; they may not even be at the same level between individual nodes in the same cluster. The hardware evaluation process should include determination of the desired BIOS revision and CMOS parameter settings. Once the proper BIOS level and CMOS settings have been determined, it should be the integrator's responsibility to ensure that these settings are correct on all nodes. Configuration of peripheral firmware (e.g., hardware RAID controllers, FibreChannel adapters, etc.) should also be specified and addressed by the integrator. Again, it may be necessary to test options and determine appropriate settings for these devices during the hardware evaluation.

The integrator should install all nodes of the cluster with some OS image and conduct burn-in tests on the hardware. This serves to address infant mortality as well as any serious hardware misconfigurations. Depending on the size of the system, the RFP may also specify minimum performance levels that the integrator should demonstrate on the system.

4.3 Hardware Support Strategy

There are a number of differences between COTS clusters and vendor-proprietary clusters that should factor into hardware support strategy decisions. Most notable among these is the cost of the hardware. COTS components such as motherboards, CPUs, memory, and hard drives are relatively inexpensive and readily available. However, on-site hardware support from a vendor or third-party maintenance supplier can be very expensive, offsetting some of the price appeal of these systems. On proprietary systems, for which the hardware can be very expensive, the cost of on-site support is often offset by the savings on parts, which are typically included in hardware support contracts. Another factor is the complexity of the repair operations. With the possible exception of motherboards, replacement of failed components in COTS clusters is fairly straightforward once the failed component is identified. Thus, even for production installations with high availability requirements, do-it-yourself hardware maintenance can be a cost-effective and reliable strategy.

LC has adopted a model that makes use of a “cold” parts cache, a “hot spare” cluster, and appropriately skilled staff members. When a new cluster is purchased, a number of additional components are procured and stored on a shelf. The quantity of each component is based on published MTBF data as well as locally collected data on similar hardware. Other factors in sizing the parts cache include the manufacturer’s warranty for each component and the tolerance level for substitution of non-identical components. If, in a year, the cache of 2.4-GHz processors runs out and 3.0-GHz processors have to be substituted in some nodes, how will this newly introduced heterogeneity affect the users of the system? If there is a need to maintain a homogeneous cluster, then more spare parts may be purchased, although this should be weighed against the risk of being left with a cache of unused, obsolete parts. Another option is the “cluster eats itself” approach in which nodes are turned off and used for spare parts once the original parts cache has been depleted.

A hot spare cluster serves a number of important purposes. It provides the quickest means for getting a failed node back into service. When the cause of failure is not obvious or the repair is known to be time consuming, a failed node can quickly be swapped with one from the hot spare cluster. A hot spare cluster can be used to burn in hardware before it is placed in the production cluster, thus preventing replacing defective hardware with new defective hardware. When the cause of failure has not yet been isolated, the problem determination can be performed off-line in the hot spare cluster, without further impact on the production resource. When a production cluster is too small to justify the purchase of a hot spare cluster, it is still a good idea to purchase an extra node or two that can be kept on a shelf and brought into service quickly in the event that a production node cannot be easily repaired. These extra nodes also ensure that there are spares of *every* part on hand.

Commodity Linux cluster hardware is similar to the PC hardware in homes and offices; therefore, finding staff members capable of repairing these nodes is not difficult. In fact, it can be difficult to *prevent* highly trained, highly paid computer scientists from running to the machine room with a screwdriver at every opportunity! The LC Operations staff performs the repairs on the Linux cluster nodes. The primary role of the LC Operations staff is to monitor production systems in real time; thus, they typically are the first to see failures. Upon noticing a failure, the operator will confer with the responsible system administrator to determine the cause of failure and the appropriate repair action (replace component in place or perform node swap). The operator will perform the repair, then notify the system administrator for followup action (validation of repair, OS reinstallation if required, etc.). A designated member of the operations staff generates return authorization requests for failed components and tracks the replacement of those components.

LC’s newest cluster from IBM, ALC (960-nodes), does include a 3-year warranty with next day on-site hardware repair. This warranty is more a luxury than a necessity. This repair contract is used to augment LC’s support strategy. Failed nodes are swapped from ALC into its hot spare cluster. A service call is then placed with IBM, who dispatches a technician to make the repair the following day.

5 System Software

Like many sites operating Linux clusters, Livermore draws from several sources to build usable system software, including the open source community, industrial partnerships, and in-house development. It was recognized early on that the lack of integration testing and release discipline in this approach would be a weakness, so after an unsuccessful search for an existing HPC Linux distribution that could serve Livermore's needs, LC began producing CHAOS, its in-house, Red Hat-based Linux distribution for clusters. CHAOS is described in Section 5.1.

CHAOS tracks Red Hat "boxed set" releases and adds HPC enhancements in the following areas: kernel (Section 5.2), cluster system management tools (Section 5.3), cluster monitoring and failure detection (Section 5.4), resource management (Section 5.5), authentication and access control (Section 5.6), and parallel file system (Section 5.7).

Because CHAOS components are gathered from multiple sources and integrated on site, Livermore owns the first level of software support. Some problems are addressed by local software developers, and others are handled by external sources, such as Red Hat Inc., Quadrics Ltd., or Cluster File Systems, Inc. The CHAOS support strategy is described more fully in Section 5.9.

5.1 CHAOS

CHAOS is currently used on all LC Linux clusters. The CHAOS strategy is to leverage the Red Hat relationship for the majority of software components that are not Livermore or cluster specific. CHAOS developers focus on cluster-specific components and on integration testing aimed at LC systems and customer workloads.

Each Red Hat "boxed set" release,[10] which occurs about every 6 months, is normally followed by a CHAOS release. CHAOS only includes a subset of the Red Hat packages that would normally be installed on a standard Red Hat desktop or server configuration. Packages are chosen for inclusion in CHAOS on the basis of user and system requirements; for example, multimedia oriented tools are not included, except as requested by visualization customers.

On top of this scaled-back Red Hat base, CHAOS adds packages that come from other sources, such as a custom kernel (Red Hat-derived), cluster management tools, cluster monitoring tools, resource manager, access control facilities, parallel file system, and application development tools. Because CHAOS contains

[10] Red Hat also offers the Advanced Server product, which, because of its 18-month release cycle, was deemed insufficiently agile to support new cluster hardware. New hardware is often incorporated into LC clusters shortly after becoming generally available.

only the exact tools needed to support the LC Linux environment at the time of release (no effort is made to retain backward compatibility with hardware no longer at LC or to support tools or operating modes not in use at LC), the CHAOS effort is well focused.

New releases of CHAOS are planned, staged, tested, packaged, and released under the direction of a release manager who ensures that each version of CHAOS is ready to be installed on production systems with no surprises and that customers are made aware in advance of any impacts. Development and testing is carried out on LC's I/O Testbed facility, which is populated with small clusters and hardware representative of the current mix of hardware running in production. Tests include regression tests for past problems as well as applications typical of a production workload. When a release is vetted on the testbed systems and is deemed production worthy, it is installed on progressively larger production clusters, so that any problems missed by testing, such as problems of scale, are discovered and solved with the least impact.

As mentioned above, the additional effort needed to integrate, test, and release CHAOS is minimal because the effort is focused. In addition, the cost of the effort is offset by the fact that LC has exclusive control over release schedules and content, which can be tailored to fit LC customer needs and production roll-out schedules for new clusters. LC does not plan to turn CHAOS into an open source project, which would imply distributed decision making. Nearly all of its components, however, are or will be released individually under the GNU General Public License (http://www.gnu.org/copyleft/gpl.html) and made available for download from the Linux@Livermore website (http://www.llnl.gov/linux/).

5.2 CHAOS Kernel

The CHAOS kernel actively tracks Red Hat errata kernel releases. Its modifications include enhanced device support and other changes such as increased resource limits needed in LC's environment. Kernel source code is managed in the Concurrent Versions System (CVS) source code control system. Kernel crash dump support and serial console logging enhance LC's ability to support the Linux kernel.

The Red Hat kernel series was chosen as the basis for CHAOS kernels over the stable Linux kernel releases (ftp://ftp.kernel.org) for several reasons. First, the kernel must be synchronized with other Red Hat components; for example, Red Hat 9 (http://www.redhat.com/docs/manuals/linux/RHL-9-Manual/release-notes/x86/) introduced the *Native POSIX Thread Library (NPTL)*, which includes substantial kernel and GLIBC changes that must be deployed together. Second, using Red Hat kernels enables LC to leverage a support relationship with Red Hat kernel developers, which has proved to be more effective than interacting directly with the Linux kernel community on problem resolution and propagation of changes upstream. Finally, the content of the Red Hat kernel is somewhat more closely aligned to Livermore needs than are the stable kernel releases. For example, LC believes that kernel crash dump support is required in a production environment, and recently Red Hat has begun to ship kernels with netdump[8] support. While not quite ready for use on clusters, netdump is a promising replacement for the aging Mission Critical Core Dump patch currently used at Livermore. In contrast, no crash dump facility is currently planned for the stable kernel series (http://lwn.net/Articles/14793/).

CHAOS kernel changes are summarized in Table 4. The kernel is maintained under CVS source code control, with scripts on hand to build Red Hat Package Manager (RPM) releases directly from CVS release tags. When a new Red Hat errata kernel is released, it is checked into a Red Hat CVS branch by an on-site Red Hat engineer and scrutinized for differences from the last Red Hat release. When the changes are understood, the new errata is merged onto the LLNL branch. Other changes are added directly to the LLNL branch. When a kernel is ready for release, it is tagged and built as an RPM. Before a release is installed on production systems, it is tested with the Cerebus (http://sourceforge.net/projects/va-ctcs/) test suite and a scaled-down version of the CHAOS test suite, which includes user applications that exercise the message passing interconnect.

Table 4. CHAOS 2.4.18 kernel modifications

Patch	*Description*
	Enhanced Device Support
QsNet Elan	Quadrics Elan3 adapter and related drivers (added)
QLogic qla2300	QLogic 2 gigabit FibreChannel adapter (updated)
MTD ich2rom	Intel firmware hub flash device (updated)
IBMASM	IBM Remote Supervisor Adapter (RSA) (added)
E1000	Intel EEPro1000 GigE adapter (updated)
AceNIC	Netgear GA620 GigE adapter (updated)
	Other Modifications
MCORE/BOOTIMG	Mission Critical Linux Crash Dump
QsNet	Adapter MMU sync, exit handlers, core file naming
TV ptrace	Ptrace changes for TotalView parallel debugger
PerfCtr	Hardware performance counter support
ECC	Poll memory errors and count/panic
p4therm	Module to detect Xeon thermal throttling and panic the node
Lustre support	Vfs intents, RO devices, zero-copy TCP enhancement (not Lustre itself)
statfs64	Support for statfs64 system call
NFS groups $>$ 16	Membership in $>$ 16 groups on NFS file systems
NFS zombie	Bug fix for unkillable zombies blocked in NFS I/O
dsp stack trace	Use frame pointers to improve oops stack traces
FDs $>$ 1024	Allow 8192 file descriptors per process

The crash dump facility, syslog, and the serial console infrastructure make possible the gathering of detailed forensics after a crash. Problems with interconnect drivers or other CHAOS-specific changes are handled locally or with the help of collaborators. Problems that appear to be generic to the Red Hat kernel are redirected to Red Hat. An on-site Red Hat engineer acts as the liaison between Red Hat engineering and the CHAOS team; it is the engineer's responsibility to ensure that problems assigned to Red Hat are indeed Red Hat problems and not problems introduced locally. When a fix is generated, depending on its urgency, it may go on both the main LLNL branch and a production release branch to allow the fix to go into production without introducing untested new functionality from the LLNL branch. Fixes that are not CHAOS-specific are sometimes applied locally and then removed when the fix appears in a Red Hat errata release.

5.3 Cluster System Management Tools

A good set of cluster system management tools is essential to the successful operation of large clusters. This is true regardless of whether the underlying OS is Linux or not; indeed, many of the tools used on the LC Linux clusters are the product of lessons learned on large IBM SP systems. The CHAOS cluster system management tool set, described below, has been successful in meeting the needs of LC Linux system administrators and has enabled systems the size of MCR to be managed with one full-time system administrator.[11] The close relationship between LC system administrators and developers has resulted in refinement of the tool set to be well matched to LC needs. The tool set includes a cluster installer, parallel remote shell tool, serial console management utility, remote power control utility, a configuration database, and static route manipulation script, all described below.

YACI (Yet Another Cluster Installer) (http://www.llnl.gov/linux/yaci/) uses a technique similar to VA System Imager to build node images in a chrooted environment, then boots a standalone OS over the network on each node, copies images to the nodes using either an NFS pull or a multicast push and installs the local disk. The image built for CHAOS includes CHAOS-specific localizations that occur after the node is installed to enable it to join the cluster with up-to-date software and configuration files. Once the management node is installed and configured with YACI (including creating the node images to install), YACI can perform the install on the remaining 1152 nodes of MCR in about 50 minutes.

Pdsh (Parallel Distributed Shell)[12] (http://www.llnl.gov/linux/pdsh/) is a high-performance, parallel remote shell utility. It has built-in, thread-safe clients for various "rcmd" implementations, including Kerberos IV and Berkeley remote shell, and can call ssh externally (but with reduced performance). Pdsh uses a "sliding window" parallel algorithm to conserve socket resources on the initiating node and to allow progress to continue while timeouts occur on some connections. Using

[11] In the LC environment, a system has one primary system administrator, but system administrators are part of a team and can rely on other team members' expertise in specialized areas. Also, the role of an LC system administrator is somewhat specialized; for example, a customer support team fields customer requests, a DPCS team manages the batch system, etc.

[12] Pdsh is similar to dsh, part of the IBM PSSP software offering, but offers improved performance and handling of error conditions.

pdsh, a system administrator can run commands in parallel across an entire cluster and can interpret the standard output of those commands using a companion script called “dshbak.” Pdsh can run a simple command like “hostname” across all 1154 nodes of MCR in less than 2 seconds.

ConMan (http://www.llnl.gov/linux/conman/) is a serial console management program designed to support a large number of console devices and simultaneous users. It supports local serial devices and remote terminal servers (via the Telnet protocol). On LC clusters, ConMan runs on a single management node. On systems with dual management nodes, the serial consoles are cross-wired with instances of ConMan running on each. ConMan maintains a persistent connection for each console it manages[13] for continuous logging. In addition to logging, ConMan supports interactive access to consoles and can be configured so that consoles are referenced on the command line using their host names. It allows the generation of a serial “break” over both Telnet and serial interfaces, which enables the kernel SysReq feature to be used to gather information on hung systems, force crash dumps, etc. Interactive users can also initiate a power cycle via configurable external commands with a few keystrokes. On LC systems, ConMan is configured to run the PowerMan client (see below) when an interactive user requests a power cycle. Interactive sessions can be shared or stolen, and a broadcast mode is supported.

PowerMan (http://www.llnl.gov/linux/powerman/) is a tool for controlling multiple RPC devices in parallel via a command-line interface. PowerMan can be configured to associate host names with specific plugs on the RPC devices, allowing the PowerMan commands to use node host names. Basic functionality consists of “power on,” “power off,” and “power status” commands. Where supported by the hardware, PowerMan can be configured to control beacons that flag nodes requiring service and to gather out-of-band temperature data. On LC clusters, PowerMan runs on a single management node. On systems with dual management nodes, PowerMan is configured so that one management node controls the other’s power and vice-versa. PowerMan holds open persistent connections to each RPC and uses an Expect-like scripting language to send commands to the RPCs and process responses in parallel. Table 5 summarizes the growing list of supported RPC hardware. PowerMan can query the power status of all plugs on the MCR system, which includes 121 ICE Box devices, in about 1 second and can power cycle the entire cluster in under 10 seconds.

Genders (http://www.llnl.gov/linux/genders/) is a collection of utilities used as a basis for cluster configuration management. The utilities operate on a table of node names and attributes and include a query tool, an rdist Distfile preprocessor, and a C and Perl application programming interface (API). When used for cluster configuration management, the genders file (typically /etc/genders) is replicated on every node of the cluster. It describes the layout of the cluster; specifically, it describes the subtle differences between nodes that need to be sensed by scripts

[13] On large systems, the default Linux limit of 1024 file descriptors per process may be inadequate, so on CHAOS kernels this limit has been raised to 8192.

Table 5. RPC devices supported by PowerMan 1.0.17

Device	*Rack Space*	*Plugs*	*Supply*
LNXI ICE Box v2	0U/rear-mount	10	2x15A/110 V
LNXI ICE Box v3	0U/rear-mount	10	2x15A/110 V
BayTech RPC3-20	1U	8	1x20A/110 V
WTI NPS-115	1.5U	8	2x15A/110 V
APC MasterSwitch Plus	1U	8	1x15A/110 V
WTI RPS-10 HD	stand-alone	1-8 units/RS485 chain	1x15A/unit
IBM xSeries RSA	0U/inside node	1-23 nodes/RS485 chain	n/a
Rackable Phantom	0U/inside node	n/a	n/a

and that are used to determine what variations of configuration files belong on a node. By abstracting this information into a plain text file, it becomes possible to change the cluster configuration by modifying only one file.

Nodeattr, the genders query tool, can be called on to provide a list of nodes that have a particular attribute. It is commonly called from system administration scripts to test whether a node has a particular characteristic before performing some operation (for example, nodes with the "qla2200" attribute might need to modprobe the qla2200 kernel module in the rc.local file, while others do not).

Dist2, the rdist Distfile preprocessor, expands specially formatted rdist macros with embedded genders attributes into node lists. When the genders file changes, dist2 can simply be rerun to redistribute appropriate configuration file variations. dist2 also facilitates rapid "localization" of nodes rejoining the cluster after a fresh operating system install or a long absence.

C and Perl APIs are provided for querying the genders file from system administration scripts and utilities that must manipulate lists of nodes or attributes and that, for performance or other reasons, cannot do so via repeated calls to the nodeattr utility. Pdsh has an option which exploits this API to target a command to a set of nodes with a common genders attribute.

Croute is a Perl script that sets static routes on the basis of the IP addresses of locally configured interfaces and information in a config file. Large clusters with multiple networks sometimes need complex network routing schemes to achieve load balancing of traffic across multiple interfaces. The croute tool provides a mechanism for expressing these routing schemes in a single config file.

5.4 Monitoring and Failure Detection

Detecting hardware and software failures and acting on them quickly is crucial to ensuring that a cluster functions properly and delivers reliable service to customers. LC clusters employ system monitoring, kernel level fault detection, and boot-time checks to detect failures, which are then handled by an around-the-clock operations staff and system administrators.

Clusters are monitored using a locally built SNMP infrastructure. The management node periodically runs HM, a multithreaded program that polls an SNMP agent on each node and stores the results in a MySQL (http://www.mysql.com/) database, where it is retrieved and displayed on a web page using PHP scripts. Problems are flagged in red on the web page. The pages for all production clusters are monitored by operations staff who can take appropriate action, such as deconfiguring faulty compute nodes to prevent allocation by the resource manager or paging the on-call system administrator. On each node, a modified UCD SNMP daemon (http://net-snmp.sourceforge.net/) is run that includes support for reading motherboard sensors via the LM_sensors (http://secure.netroedge.com/∼lm78/) package, as well as other functions that are compiled into the SNMP daemon to minimize the impact of each polling cycle on applications. Table 6 lists most of the entities currently monitored using SNMP.

Table 6. Entities monitored by SNMP

Entity	*Description*
crashdumps	Crash dumps in /var/dumps
daemons	Critical daemons
ECC	Correctable memory error counts
filesystems	Local file systems percent free
load	System load average
neterrs	Network interface errors
nfs	NFS file systems mounted
ntp	NTP time synchronization
rms	Status of RMS partitions
orphans	Orphaned application processes
sensors	Fan speeds, power supply voltages
swap	Available swap space

Ganglia (http://ganglia.sourceforge.net/) is also run continuously to provide a method, in addition to SNMP (which is less frequently updated), of detecting when nodes have crashed or locked up such that they can no longer multicast a heartbeat on the management network. A library exports this information to other tools, such as pdsh, which optionally use it to skip over nodes that would otherwise time out. In addition, a command-line tool can display a quick summary of up/down nodes for system administrators. Ganglia has been configured so that only the management nodes listen to heartbeat messages and has been modified to hold back other messages it would normally send. This minimizes application disruption.

On a cluster running computations where the answers matter, an uncorrectable memory error should result in a kernel panic. The CHAOS kernel has been modified to detect memory ECC errors using the ECC module (http://www.anime.net/∼goemon/linux-ecc/) for chip sets that LC has deployed. A /proc file allows SNMP to obtain and report correctable error counts by memory module, which, if excessive, indicate that memory should be proactively replaced. When an uncorrectable error occurs, the kernel panics, calling out the offending memory module. This effectively removes the node from play and terminates any running application that may have been affected by bad data.

The Linux kernel by default generates an “oops” message on its console and continues on when it dereferences a bad pointer. Because this results from a kernel bug that should be detected and tracked down, and can cause data corruption, a better result is to have the kernel panic. The Mission Critical Core Dump patch employed in CHAOS kernels provides a “panic on oops” sysctl (kernel tuning) parameter which achieves this result. Panic on oops is always set on LC Linux clusters.

Another kernel-level fault detection mechanism is a module that polls the Xeon IA-32_THERM_STATUS Model Specific Register (MSR)[9] and panics the node when the Xeon thermal sensor is tripped. In the absence of this module, an overheating node can silently begin to run at a reduced system clock duty cycle. In most parallel applications run by LC customers, a slow node may have a substantial performance impact.

When a node boots, a script called “checknode” runs through a sequence of hardware checks to ensure that the node boots up with the expected amount of memory, number of CPUs, CPU clock speed, etc. Checknode uses genders (discussed in Section 5.3) to retrieve the expected values for each node. If checknode finds any problems, it writes a description of the problem to a file (/etc/badnode), which subsequent startup scripts will abort upon finding, thus preventing the node from being brought on-line. A node in this state will be flagged by the SNMP monitoring system.

5.5 Resource Management

LC provides a common batch scheduling and resource accounting system across all its platforms called DPCS (Distributed Production Control System) (http://www.llnl.gov/icc/lc/dpcs/dpcs_overview.html). DPCS interfaces with the native resource manager on each platform. On capability Linux systems, the resource manager is currently RMS from Quadrics, although a transition to SLURM is under way. On capacity systems, DPCS interacts with an instance of the NQS (http://asis.web.cern.ch/asis/products/CERN/nqs.html) batch queuing system running on every node. LC’s resource management strategy provides flexibility that enables it to handle the unique requirements that arise in a dynamic environment.

DPCS provides a common user interface for batch job submission and accounts for resource utilization center wide. It is a “metabatch” system in the sense that users can request execution on the first cluster that has resources available that meet a list of constraints. DPCS implements a fair share scheduling policy using banks to account for resources consumed by users and projects. Reports generated from DPCS show how well LC clusters are being utilized and by which groups; these reports are invaluable in planning future procurements.

DPCS relies on a native resource manager to obtain resource availability information, to allocate resources, and to shepherd jobs through the system. It interacts with the resource manager through APIs provided by the resource manager and shell scripts. DPCS also has a daemon on every node that can monitor the resource consumption for accounting purposes.

On Linux capability clusters, RMS is currently the resource manager. RMS runs on large systems, such as ASCI Q (http://www.llnl.gov/asci/platforms/lanl_q/) under the Tru64/Sierra product, and its scalability has therefore been tested; indeed, it performs well on Linux clusters the size of MCR at 1154 nodes. RMS's proprietary licensing is, however, somewhat at odds with LC's open source Linux strategy, and it assumes the QsNet interconnect. Because of these limitations, LC initiated an effort to create an open source resource manager, SLURM, which will be released under the GPL and is designed to be interconnect-independent. SLURM is running on LC I/O Testbed systems, with plans for roll-out on ALC in the near future.

On capacity clusters, DPCS relies on NQS to shepherd jobs. In contrast to capability clusters where whole nodes are usually allocated to jobs, capacity clusters run multiple jobs per node. DPCS monitors resource utilization in real time and uses this information to make scheduling decisions. Although it is difficult to conclusively determine the future resource needs of running jobs, this method does achieve high utilization of capacity resources and in practice rarely oversubscribes the nodes.

Some capability systems pose unique resource management challenges. These challenges can be addressed because LC controls the development of DPCS and SLURM. For example, compute nodes on some visualization systems are equipped with specialized hardware that factors into scheduling decisions. Some nodes may be connected to a Power Wall, some may be connected to user display devices, and others may have specialized rendering hardware. Visualization applications requiring a mixture of these capabilities (such as a specific Power Wall and some number of rendering nodes) need sophisticated resource management algorithms. This particular problem is being addressed in SLURM and DPCS.

5.6 Authentication and Access Control

Users authenticate to cluster login nodes using LC's One-Time Password (OTP) system, or if on-site, with passwordless ssh (via personal RSA keys). Authentication and access control policy are managed on Linux using PAM, which restricts access to the various node types within the cluster based on group membership and resource manager allocations. Account and password management occurs outside the cluster, and authentication-related files are propagated between cluster nodes using rdist. Within the cluster, users may use the Berkeley r-commands (rsh, rlogin, rcp) without passwords between nodes, subject to PAM-enforced access restrictions. R-command authentication will soon be replaced with MUNGE, a scalable intra-cluster authentication service.

A “pam_otp” module exists in the PAM stack of login nodes to allow users to authenticate using an LC OTP password when coming in via ssh. The ssh configuration forces users to authenticate through the PAM stack (and therefore type their OTP) when originating off-site; otherwise, they may authenticate using their personal RSA keys without typing a password. The authentication policy for other service nodes is the same, except that access is restricted via the “pam_access” module to members of a particular UNIX group populated with support team members. All external access to the cluster is with ssh.

On capability clusters, a PAM module called “pam_rms” (http://www.llnl.gov/linux/pam_rms) allows users to log in to compute nodes RMS has allocated to them and check the progress of long-running applications, while prohibiting users from hijacking resources on compute nodes not allocated to them. A similar strategy is used by SLURM. System support staff are exempted from this restriction by the “pam_access” module.

Password and group files are managed externally by the LC Hotline. A periodic export from the Hotline’s account database is used to regenerate the “user” portion of a master password and group file on the management node. The “system” portions of these files, which contain local passwords for root accounts and other accounts needed by software subsystems, are managed using native Linux commands on the management node. The master copies are propagated to the rest of the nodes periodically using rdist.

PAM is configured to enable the Berkeley r-commands to work intra-cluster without passwords, subject to access restrictions described above. The r-commands, used by pdsh (and in some cases by RMS) to remotely execute concurrent processes, are more attractive for this use than sophisticated remote shell facilities such as ssh, because their lightweight protocol enhances performance. They are, however, built on an aging, unsafe code base, they have to be setuid root to obtain access to a privileged socket, they are prone to various well-known attacks, and they are limited in scalability because each connection requires a privileged socket.[14] The MUNGE scalable authentication service, currently nearing completion, authenticates users to services using a lightweight protocol involving a shared secret known to root on each node. MUNGE will become the authentication scheme of choice between cluster nodes and will be used by the “m-commands,” drop-in replacements for the Berkeley r-commands that use MUNGE authentication, and by SLURM.

5.7 Parallel File Systems

A parallel file system is required on capability systems that implement the Livermore Model (described in Section 2). The file system must provide a POSIX

[14] Privileged sockets are in the range of 0 to 1023, and the rresvport function used by the r-commands allocates reserved sockets in the 512-1023 range. Because each rsh command requires two sockets, one for stdin/stdout, one for stderr/control, at most 256 simultaneous rsh commands can run on a system, assuming no other r-commands are running.

interface, perform well, and be stable at all times. LC's file system strategy is focused on Lustre.

LC parallel file systems, which include IBM's GPFS on SP systems and HP's PFS on Sierra systems, are normally mounted on all compute and login nodes within a cluster. They are used to store large files such as application data files and restart dumps. Because of the size of these data sets, which can be multiple terabytes, the onus of backups is placed on users, who are provided with a high-speed communication path from the cluster to the LC storage archive and visualization systems.

Parallel file systems achieve performance targets by striping data across multiple I/O servers, each of which has access to some high-performance storage such as RAID devices on a FibreChannel SAN, local ATA RAID controllers, or NAS devices. The striping allows the file system to deliver performance to a parallel application that is not bounded by the I/O performance of a single server node, a single client, or a single network adapter. Ideally, performance would scale linearly until a maximum aggregate performance is reached and then "flat line" at the maximum.

There are several parallel file systems currently available for Linux, including PvFS (http://parlweb.parl.clemson.edu/pvfs/) from Clemson University (which performs well but suffers from design limitations that inhibit its scalability), GFS (http://www.sistina.com/products_gfs.htm) from Sistina (which assumes all nodes mounting the file system have SAN access to the underlying storage), and GPFS (http://www-1.ibm.com/servers/eserver/clusters/software/gpfs.html) from IBM (which is not open source and is only supported on IBM hardware). Because of these limitations, and also because a partnership was being sought that could enable other LC goals, such as building a distributed, parallel file system that could be used across the entire data center, LC did not attempt to join any of the existing Linux parallel file system efforts.

In April 2001 an ASCI PathForward RFP for an SGS (scalable, global, secure) file system led to the signing of a contract[10] with Cluster File Systems, Inc. to develop Lustre, an open source, distributed, object-based file system with a design informed by the Intermezzo, Coda, and GPFS file systems. The first major Lustre deliverable is Lustre Lite, an implementation with less scalability and lower meta data replication requirements, suitable for intra-cluster use.

Lustre uses OSTs to implement its object storage protocol. LLNL has partnered with BlueArc Corporation and Data Direct Networks to build storage appliances that embed the OST protocol. A Linux cluster node with direct-attached storage may also serve as an OST. During the development of Lustre, both BlueArc OSTs and Linux-hosted OSTs directly attached to DDN disks are being used for testing.

Lustre Lite nears production readiness and is currently being tested at scale on MCR using BlueArc OSTs and on ALC using Linux OSTs. At the time of this writing, a production deployment schedule for Lustre Lite is being worked out.

5.8 Application Development Environment

As described previously, the Livermore Model seeks to provide a common application programming framework across all LC production systems. Different OSs and architectures may constrain this goal, in which case the focus shifts to providing the required functionality even if the actual tools differ from platform to platform.

Table 7 highlights some of the tools that are available on the LC Linux clusters. LC's Development Environment Group (DEG) is tasked with identifying, deploying, and supporting the tools that comprise this common application programming framework. An in-depth discussion of these tools is beyond the scope of this paper.

Table 7. Application development tools

Tool	*Description*	*Vendor*
Intel compilers	C, C++, Fortran compilers (preferred)	Intel
PGI compilers	C, C++, Fortran compilers	PGI
GNU compilers	C, C++, Fortran compilers	Open source
Quadrics MPI	MPI derived from MPICH 1.2.4	Quadrics (Open source)
TotalView	Debugger	Etnus
PAPI	Hardware counter tool	U of Tenessee (Open source)
Valgrind	Memory correctness tool	Open source
VGV, Vampirtrace	Parallel profiling tool	KAI (Intel), Pallas
MKL	Math Kernel Library	Intel

5.9 Software Support Strategy

The CHAOS support strategy is for system administrators, local developers, and industrial partners to work together to resolve software problems. Trouble tickets and software defects are tracked with automated systems. Software fixes of an urgent nature may be installed in production, although bug fixes are deferred until a future CHAOS release whenever possible to avoid unnecessary drift between releases. When appropriate, fixes are pushed back to the authors of software to minimize the number of patches that have to be maintained locally.

When a problem is encountered on a production system (typically reported by an end user or system administrator), the system administrator performs the initial problem determination and attempts to fix the problem. If the problem stems from a software defect, a trouble ticket is generated for the CHAOS development team for further analysis. If a developer verifies that the problem is indeed in the software, a bug report is entered into the project defect tracking system. The development team pursues root cause analysis, interacting with the system administrator and/or end user to assist in problem isolation and recreation if necessary. When possible, an attempt is made to recreate the bug on one of the testbed systems, allowing ongoing debugging and testing of candidate fixes to proceed without affecting the production systems.

If the problem is in software that is maintained by industrial partners (e.g., in software distributed by Red Hat or Quadrics), the CHAOS developers engage these partners in pursuing a fix. At this stage, the problem is often fairly well characterized, and a simple “reproducer” is available.

Every effort is made to avoid allowing untested functionality to be installed in production. When a fix is available, a decision is made as to whether there is an urgent need for the fix to be pushed out immediately. Bugs with significant impact on system security, stability, usability and/or performance might qualify; otherwise, fixes are deferred to the next CHAOS release. If urgent, then the fix is usually made against the version currently running in production as well as against the current development version, with branches created in the CVS source code control system as appropriate. An RPM is built for the fixed software, which is tested on the I/O Testbed facility. Testing may consist of rerunning the entire CHAOS regression suite or simple unit testing, depending on the nature of the bug and the relationship of the affected software to the rest of CHAOS. Once validated, the RPM is installed in production.

Minimizing the number and scope of local modifications is a key element of the supportability of CHAOS. Each local modification must be reapplied to each new release, which introduces more opportunities for a patch/modification to be accidentally reverted or broken. When defects are discovered in software owned by Red Hat or other collaborators, the fixes are, when possible, pushed “upstream” with the hope that they will appear in a subsequent release and no longer need to be reapplied.

6 Lessons Learned

While the deployment of these Linux clusters within LC has been a success, it was not without learning some lessons. Most of these lessons have been learned the hard way, but they provide valuable insight on how to build and manage large HPC Linux clusters.

It is important to communicate even the most basic aspects of the desired system design and integration requirements to the vendor and avoid making assumptions about how they will assemble the cluster hardware. When integrating a 70-node cluster, about one-third of the nodes failed to install the OS image. The failure was eventually determined to be caused by miscabling of the IDE devices in the nodes. Some nodes correctly had the hard drive on IDE channel 0 and the CD-ROM drive on IDE channel 1; other nodes had this reversed. Thus, when the installation software attempted to install an OS image on the hda partition, on some nodes this was the CD-ROM drive instead of the hard drive. It is difficult to specify system requirements clearly enough to avoid problems during the integration; however, with careful consideration and clearly stated expectations, problems can be minimized, if not prevented entirely.

During the initial testing of LC's first production HPC Linux clusters, some nodes produced wrong answers. After much experimentation and testing, the problem was isolated to a subset of nodes, and the primary cause of the problem determined to be uncorrectable memory errors. On some motherboards, Linux ignores these errors and continues. Understanding these errors led to the inclusion and planning for ECC error detection on current and future systems.

When integrating a small, single rack cluster, an overly simplistic specification of a 19-inch rack was used. The provided rack was too shallow for the nodes that were to be racked in it. LC system administrators spent quite some time trying to make this rack work before finally giving up and locating another rack. This experience resulted in careful specifications for racks for subsequent systems.

When MCR was being integrated, some applications had unpredictable performance. Jobs were unexpectedly slow some of the time. Again, it took time to determine that, because of high processor temperatures, the Pentium Xeon processors were dropping to a reduced clock duty cycle. Following this discovery, the p4therm kernel module was developed and has been deployed on all LC production systems using Xeon processors.

After MCR was delivered, it became evident that despite careful planning by experienced facilities designers, LC had not provided adequate cooling. At one point, an engineering team was called in to diagnose a "hot spot" in MCR. Several small groups of nodes were overheating despite ample CRAC capacity. A detailed analysis revealed that air exhausing out of the tops of the racks was creating eddy currents that caused this air to be redirected to the intake of the top nodes in the racks. A number of solutions were evaluated, and the problem was eventually resolved by using the false ceiling as a return plenum to the air handlers on the perimeter of the room.

Terminal server specifications have also been a problem for LC. When ALC was being integrated, system administrators discovered that the baud rate configured on the terminal servers was causing lost and corrupted data when logging on all ports of the terminal servers. The vendor asserted that the ports could be run at 115,200 baud with flow control, but it was subsequently discovered that enabling flow control led to kernel instability when there was excessive terminal server activity. Upgrading the memory in the terminal servers improved the situation and ultimately a setting of 38,400 baud without flow control was deemed acceptable.

It is also important to ensure that Linux-based tools are available for upgrading BIOSes and changing CMOS settings for the nodes. When ALC was being installed, the integration personnel actually used DOS boot floppies to flash the BIOSes on all 960 nodes of the cluster. This included some of the nodes being flashed multiple times as the correct BIOS release was determined and necessary settings found. Needless to say, this would have been completely unacceptable if LC staff had been required to do this. Fortunately, IBM was already developing Linux-based tools for handling this, and they provided a beta release of them before LLNL needed to change CMOS settings or BIOS levels across the cluster.

These are only a few of the situations that have been encountered by LC while deploying and managing HPC Linux clusters. However, they represent a variety of the problems that can be encountered and show how LC has learned from mistakes and adapted to deal with them.

7 Conclusions

LC has successfully deployed the 11 systems listed in Table 8 using the strategies discussed above. These clusters range in size from an 8-node development cluster to the 1154-node MCR. Three CHAOS releases have been installed on these systems, with a fourth release on the way. The systems have proved to be manageable in a demanding production environment and have and a total cost of ownership (TCO) that compares favorably to that of proprietary clusters. The success of these systems demonstrates that world-class HPC compute resources can be built and maintained using COTS hardware and open source software.

Table 8. LC clusters

System	*No. Nodes*	*Vendor*	*Integrator*	*Architecture*	*Chip Set*	*Interconnect*
Production Systems						
Adelie	128+1	LNXI	SGI	Dual 1.7-GHz Xeon	Intel 860	Quadrics
Emperor	96+1	LNXI	SGI	Dual 1.7-GHz Xeon	Intel 860	Quadrics
Pengra	16+1	Acme	LLNL	Dual 2.2-GHz Xeon	Intel E7500	Quadrics
ILX	45+1	Acme	LLNL	Dual 2.4-GHz Xeon	Intel E7500	none
PVC	70+1	Acme	LLNL	Dual 2.4-GHz Xeon	Intel E7500 / Intel 860	Quadrics
MCR	1152+2	LNXI	LNXI	Dual 2.4-GHz Xeon	Intel E7500	Quadrics
ALC	960+2	IBM	IBM	Dual 2.4-GHz Xeon	Serverworks GCLE	Quadrics
Development Systems						
Dev	20+1	LNXI	SGI	Dual 1.7/3.0-GHz Xeon	Intel 860 / E7501	Quadrics
Mdev	26+2	LNXI	LNXI	Dual 2.2-GHz Xeon	Intel E7500	Quadrics
Adev	16+2	IBM	IBM	Dual 2.4-GHz Xeon	Serverworks GCLE	Quadrics
Toad	8+2	LNXI	LLNL	Dual 1.0-GHz PIII	Serverworks SSIII/LE	Quadrics

7.1 System Manageability

LC Linux clusters exhibit high levels of manageability, usability, and reliability. This result was achieved with careful attention to hardware choices and integration details and robust system software. The MCR cluster is a particularly good example of these successes, both in comparison to a similarly sized proprietary system, ASCI White (http://www.llnl.gov/asci/platforms/white/), and another similarly sized Linux cluster from another vendor, ALC.

ASCI White, delivered to LLNL in 2000, is a 512-node, 8192-processor IBM SP cluster running AIX and the Parallel System Support Program (PSSP) software stack for cluster system management. Although the node and processor counts are not identical to the 1154-node, 2308-processor MCR cluster, the two systems achieved similar LINPACK results and can be viewed as similarly sized systems for the purposes of this comparison.

MCR has been significantly easier to manage than ASCI White. Some of the advantages of MCR are due to better cluster management tools. For example, a parallel "hostname" command executed across all 1154 nodes of MCR using LLNL's pdsh tool takes 1.57 seconds. A similar command across all 512 nodes of ASCI White using the PSSP dsh tool takes over 5 minutes. A reboot of MCR, including mounting NFS file systems, starting the RMS batch system, and mounting Lustre file systems, takes approximately 30 minutes. A reboot of ASCI White, including starting the SP switch, mouting NFS file systems, starting the LoadLeveler batch system, and mounting GPFS file systems, takes about 5 hours.[15] A complete reinstall of MCR using YACI, including installation of the management node and image setup, takes about 80 minutes. A complete reinstall of ASCI White was recently performed, including a migration upgrade of the control workstation. This upgrade took almost one week. Because of the speed and simplicity of cluster reinstallations using YACI, major CHAOS upgrades are typically accomplished by a reinstall rather than an RPM update. A reinstall has the additional advantage of cleaning up any file system "cruft" (stray log files, core dumps, etc.) as well as eliminating accumulated local disk fragmentation.

Comparing MCR to ASCI White is perhaps an "apples to oranges" comparison given the substantial differences in both the hardware and software environment. A comparison of MCR to ALC shows how subtle hardware differences can affect overall system manageability. All COTS Linux clusters are not created equally.

ALC is a 962-node Linux cluster from IBM. MCR is an 1154-node Linux cluster from Linux NetworX. Both clusters use dual 2.4-GHz Intel Xeon processors, 4-GB DDR memory and 120-GB IDE hard drives. The IBM nodes are primarily 1U xSeries 335 servers utilizing the Serverworks GC-LE chip set on an IBM motherboard, Broadcom NICs, and IBM's BIOS. The Linux NetworX nodes are primarily 0.8U Evolocity servers using the Intel E7500 chip set on a Supermicro motherboard, Intel NICs, and LinuxBIOS. ALC uses MRV terminal servers for console management and IBM's proprietary RSA hardware for power management. MCR uses proprietary ICE Box hardware from Linux NetworX for both console and power management. Both clusters use identical Cisco network switches and the Quadrics Elan3 high-speed interconnect.

In cases where the cluster hardware is not a factor, MCR and ALC behave very similarly. A pdsh of the "hostname" command across all 962 ALC nodes takes 1.14 seconds. The time to reinstall ALC using YACI is very similar to MCR. The cluster reboot time on ALC is slightly longer due to the overhead incurred by using the vendor BIOS as opposed to LinuxBIOS on MCR, but because the reboot occurs in parallel on all nodes, this overhead is fairly negligible.

When dealing with software components that interface with the motherboard chip set, as well as with power management functions, the differences between

[15] A collaborative effort by Pacific Northwest National Laboratory, LLNL, and the National Centers for Environmental Prediction to improve SP boot time achieved a reduction of over one hour for large systems[11]. However, IBM declined to accept the proposed software changes into the PSSP product.

MCR and ALC become more noticeable and, for ALC, manageability suffers. Unavailability of chip set programming information for ALC's Serverworks GC-LE chip set means a number of important system management functions cannot be supported on this platform using existing CHAOS tools. The ECC kernel module, LM_sensors utility, and MTD flash device all require chip set specific information that is obtainable for Intel chip sets but not for Serverworks. Porting LinuxBIOS to the IBM xSeries servers is not possible for the same reason. On ALC, a BIOS setting allows a system reset to be performed in the event of an uncorrectable memory error; however, this method fails to provide a console message indicating the cause of the crash, as would be generated by the ECC module. Additionally, correctable memory errors cannot be tracked on the ALC nodes. Recently, IBM provided a set of beta tools to gather sensor data from the service processor in the xSeries nodes, as well as to allow CMOS parameter manipulation and BIOS flash operations from Linux. Although these tools appear to provide acceptable functionality, they are closed source and thus will require an ongoing reliance on IBM for support.

ALC's power management solution has also been problematic. The IBM RSA adapter implements a firmware-based web server that is used to issue power on/off commands to the nodes on the RSA chain. Each chain consists of approximately 20 nodes using RS-485 connections to the service processors in these nodes. The RSA web interface has proven to be unreliable, occasionally requiring multiple commands to power cycle a node. RSA adapters can hang, requiring a power cycle of the adapter. The onboard service processors have also been known to hang, preventing a remote power cycle and requiring a trip to the machine room to manually power cycle the affected node. LC developers have been working with IBM to develop a Telnet-based interface to the RSA adapter, which is hoped will alleviate some of these problems in the future. While the Linux NetworX ICE Box product used on MCR is more expensive on a per-node basis, it is a well-integrated solution with a straightforward interface, enabling easy integration into the CHAOS tool set (PowerMan, ConMan).

ALC has experienced a significantly higher rate of software crashes due to bugs in drivers for proprietary chip set functions. The Broadcom tg3 driver caused extensive instability in the early integration phases of ALC, although a recent driver update has shown substantial improvement. Another failure mode relates to the OSB4 driver for the Serverworks South Bridge chip that drives the IDE controller. The driver sometimes reports it is in an impossible state when the system attempts to fsck the hard disk after a system crash. When this occurs, the only possible recovery is to reinstall the node. The CHAOS development team has reviewed the OSB4 driver source code but is unable to resolve this problem without chip set documentation. Investigation by IBM is ongoing.

7.2 Total Cost of Ownership

HPC Linux clusters have a compelling TCO advantage. There is significant cost savings at time of purchase, and using the hardware and software support strategies detailed earlier in this paper keeps TCO low. Instead of signing expensive support contracts, LC has hired appropriately skilled staff to support these systems in house. This includes a staff of six system administrators and eight system developers. The Linux cluster system administrators have varied backgrounds, including Tru64 clusters, large SPs (including ASCI White), and Irix systems. The system administration staff is on par with the number and skill level needed for handling a similar number of proprietary HPC clusters (like IBM SPs). The expense of the developers is counterbalanced primarily by the lack of expensive software support contracts (or much less expensive contracts that augment the support provided by in-house developers). It is also important to note that by having in-house support, LC controls the future directions of the software used on LC systems. When dealing with a proprietary system vendor, it is possible that new feature requests, or even in some cases bug reports, will not be addressed. With in-house support of open source software, if the bug fix or feature request is a priority to LC, it will be addressed by the LC Linux development staff.

8 Future Directions

LC continues to develop open source software in a number of areas. One of the biggest needs for production deployment of large HPC clusters at LC is a reliable and scalable parallel file system. LLNL is currently partnered with Cluster File Systems, Inc. to develop Lustre Lite and Lustre. This partnership should provide parallel file systems for use on LC clusters both in the near term and well into the future. LC also continues to develop the cluster management tools, installation tools, kernel, etc., that comprise CHAOS.

HPC systems are currently encountering the limits of 32-bit microprocessors. New processor architectures, including Intel IA64 and AMD x86-64, are available, and LC is evaluating both of these for future clusters.

Interconnect technologies also continue to be a prime area of interest to LC. While QsNet Elan3 has worked very well for current clusters, LC is evaluating new interconnect technologies, including Infiniband and QsNet Elan4.

On existing LC clusters, diskfull configurations contribute to overall MTBF. Exploring and characterizing diskless cluster configurations is another area for future research.

References

1. Garlick, J.E., Dunlap, C.M.: *Linux Project Report*. Lawrence Livermore National Laboratory, CA, UCRL-ID-150021 (August 18, 2002)
2. Garlick, J.E., Dunlap, C.M.: *Building CHAOS: an Operating System for Livermore Linux Clusters*. Lawrence Livermore National Laboratory, CA, UCRL-ID-151968 (February 21, 2002)
3. Seager, M., et al.: *Multiprogrammatic and Institutional Computing Capability Resource: Statement of Work*. Lawrence Livermore National Laboratory, CA, UCRL-CR-148022 (April 19, 2002)

4. Dell, T.J.: *A White Paper on the Benefits of Chipkill-Correct ECC for PC Server Main Memory*. IBM Microelectronics Division, available from http://www.ibm.com/servers/eserver/pseries/campaigns/chipkill.pdf (Nov. 19, 1997)
5. Minnich, R., et al.: *The Linux BIOS*. In Proceedings of the 4th Annual Linux Showcase and Conference, Atlanta, available from http://www.linuxbios.org/papers/als00/linuxbios.pdf (August 15, 2000)
6. Jette, M.A., Grondona, M.A.: *SLURM: Simple Linux Utility for Resource Management*. Lawrence Livermore National Laboratory, CA, UCRL-MA-147996-REV (August 28, 2002)
7. Braam, P.J., et al.: *The Lustre Storage Architecture*. Cluster File Systems, Inc., Mountain View, CA, available from http://www.lustre.org/docs/lustre.pdf (September 17, 2002)
8. Johnson, M.K.: *RedHat, Inc.'s Network Console and Crash Dump Facility*. Red Hat, Inc., available from http://www.redhat.com/support/wpapers/redhat/netdump/ (2002)
9. Intel Corporation: *IA-32 Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*. Intel Corporation, available from http://www.intel.com/design/Pentium4/manuals/245472.htm (2002)
10. Braam, P.J., Zahir, R.: *Lustre Project Technical Summary, Attachment A to RFP B514193 Response*. Cluster File Systems, Inc., Mountain View, CA, available from http://www.lustre.org/docs/SGSRFP.pdf (July 29, 2001)
11. Studham, R.S., et al.: *Process Refinements to Reboot a Large SP*. Pacific Northwest National Laboratory, WA, PNNL-SA-34389 (2001)